Application Note:
TLT-0625-AN-MICROEJ-FromScratch

Building a Java Platform from scratch

In relation to: MICROEJ products

Features

This Application Note explains how to create a Java Platform using the MicroEJ® environment and the ST Standard Peripherals Library, and how to build it using Keil µVision.

Description

This Application Note assumes the reader wishes to understand the steps involved in creating a Java Platform. It explains in detail all the steps required to build the platform and test it with a simple application that outputs text via the board's serial port.

Table of Contents

List of Figures

1 Introduction

1.1 Intended audience

The intended audience for this Application Note are developers who wish to connect a Java Platform to an existing BSP (board support package). In this Note we use the STM32Fxxx Standard Peripherals Library as the board support package.

1.2 Scope

This Application Note describes the steps required to build a MicroEJ Java Platform for a STM3220G-EVAL or STM3240G-EVAL board "from scratch" - meaning that the required steps are performed manually rather than by one of the examples supplied with the MicroEJ environment[1].

The MicroJvm® virtual machine will execute as a task controlled by the RTX Kernel. The Java threads are managed by the MicroJvm virtual machine – they execute within the MicroJvm virtual machine task. This is often called a "green thread" configuration.

This Note does not explain the contents of the C files used to bind the MicroJvm virtual machine to the RTX Kernel and the Standard Peripherals Library. For information about the content of these files see Low Level MicroJvm in the platform Reference Manual. For information about the content of `main.c` see the SNI Specification.

1.3 Prerequisites

The MicroEJ environment for MicroEJ® (MICROEJ-PKG-STD-MicroEJ-3.1.0 or later) must be installed and any required licenses obtained.

Keil µVision version 4.54 or later must be installed, with its license.

A STM3220G-EVAL or STM3240G-EVAL board must be available, connected to the desktop PC through the ST-LINK/V2. An RS-232 cable must connect the 9-pin connector CN16 to a suitable terminal emulator so that output from the board can be viewed.

The Standard Peripherals Library appropriate for the board being used must be downloaded from the IS2T web site.

For the STM3220G-EVAL board, the appropriate library can be found at: http://www.is2t.com/download/stsw-stm32062.zip

For the STM3240G-EVAL board, the library is at: http://www.is2t.com/download/stm32f4_dsp_stdperiph_lib.zip

[1] Creating and building a "Java Platform Configuration" based on an example will typically perform the steps described in this document automatically.

2 An outline of the required steps

To create and test a Java Platform:

1. Create the Java Platform (JPF)

   A JPF is created within the MicroEJ environment.

2. Create the Java application that will test the JPF

   A Java project is created within the MicroEJ environment, and the required code written.

3. Build the Java application binary file

   The Java project is built in the MicroEJ environment, targeting the embedded JPF (“embJPF”), to produce a binary object file that can be linked by µVision.

4. Create a file structure to hold the C code

   The template project provided within the STM32Fxxx Standard Peripherals Library (SPL) is used to create a project to hold the C code for the application.

5. Add the MicroJvm virtual machine support files to the C project

   The files supplied with this Application Note are added to the C project.

6. Configure the Keil µVision project

   The properties of the µVision project supplied with the SPL are adjusted so that it includes all the required sources and libraries, and refers to the correct header files.

7. Build and deploy the C project

   The C project is built and deployed to the target board using the ST-LINK/V2 connection.

3 The steps in detail

3.1 Create the Java Platform

A Java platform (JPF) comprises the MicroJvm virtual machine itself plus supporting libraries and tools. A JPF is suitable for use on a specific core and toolchain depending on the chosen architecture.

3.1.1 Create the Java Platform configuration

The first step is to create a JPF Configuration project that will be used to parameter the JPF. In this application note, we will build a JPF compatible with Cortex-M3 MCU. Create the JPF Configuration by selecting, in the MicroEJ environment : **File** → **New** → **Java Platform**

A dialog box appears in order to choose the JPF architecture and to suggest to start from an JPF example. For the application note, select the CORTEX-M3-based JPF architecture. To create a JPF "from scratch", uncheck the option "Create a platform from an example or a template".

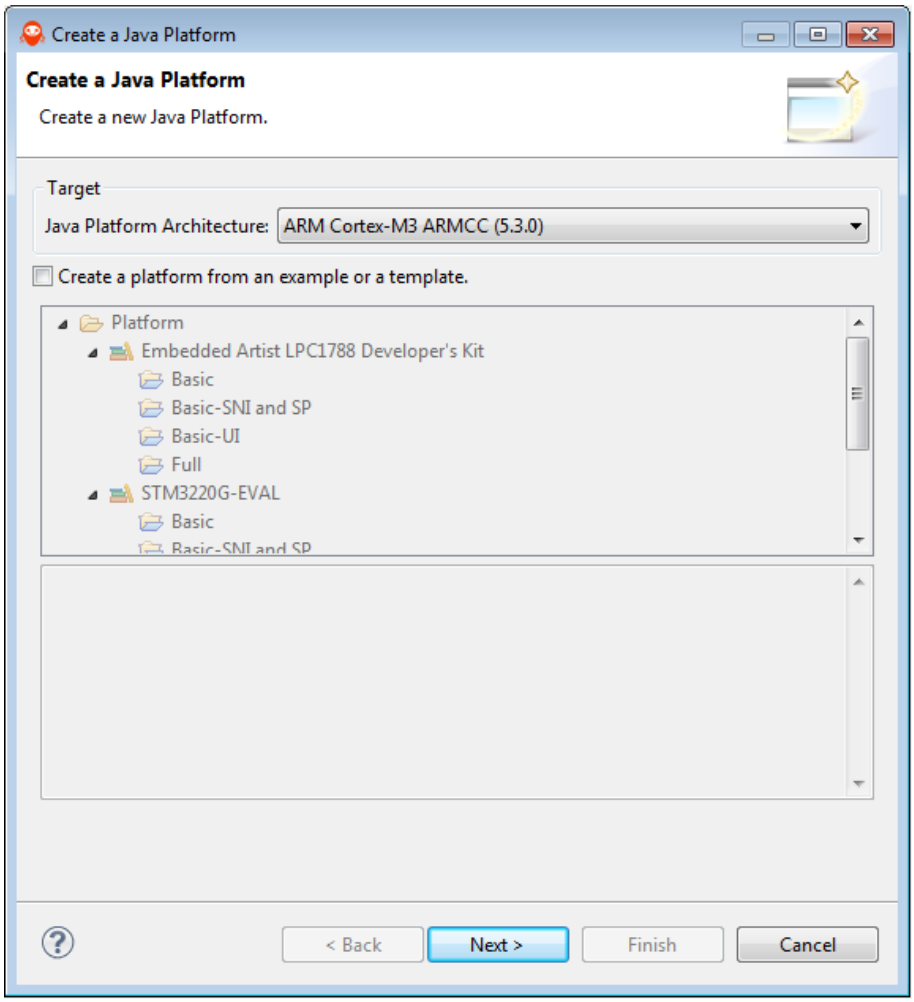

Figure 3.1. Java Platform Configuration (architecture selection)

Click on **Next**. JPF creation wizard continues asking to set a name for the future project and properties for the created JPF. Please fill the form like following:

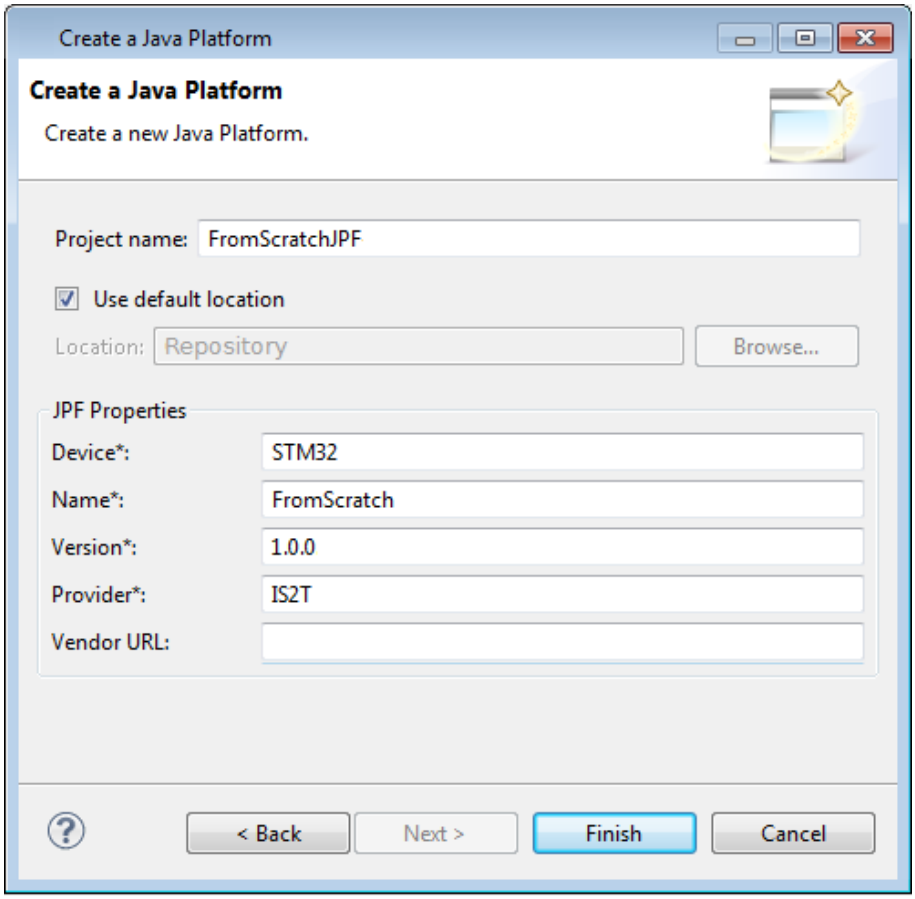

Figure 3.2. Java Platform Configuration (properties set)

The provider can be any name you wish. On pressing **Finish**, a new project is created containing the JPF configuration.

3.1.2 Build the Java Platform

The second step is to build the JPF based on the JPF Configuration. Build the JPF by opening the **FromScratch.platform** file created on the JPF Configuration project and selecting the **Build Platform** hyperlink available on right side of the panel:

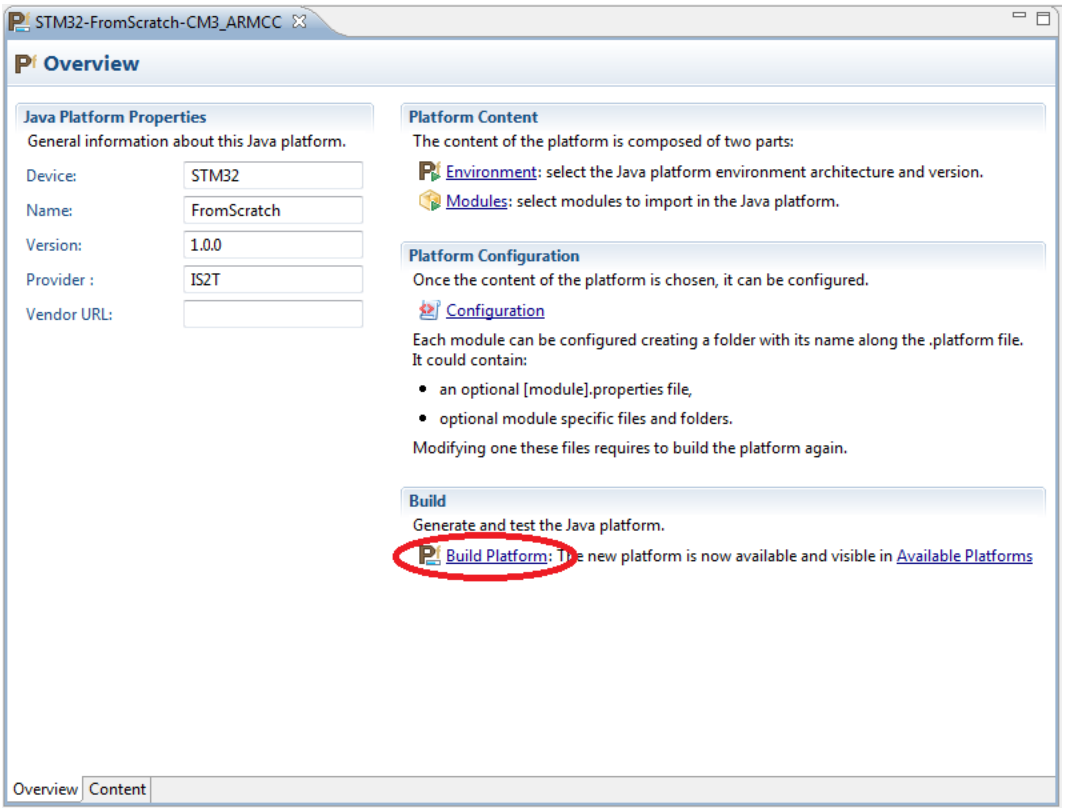

Figure 3.3. Java Platform Configuration (build platform)

The JPF name will be STM32-FromScratch-CM3_ARMCC[2]. Your workspace should look like the one shown below:

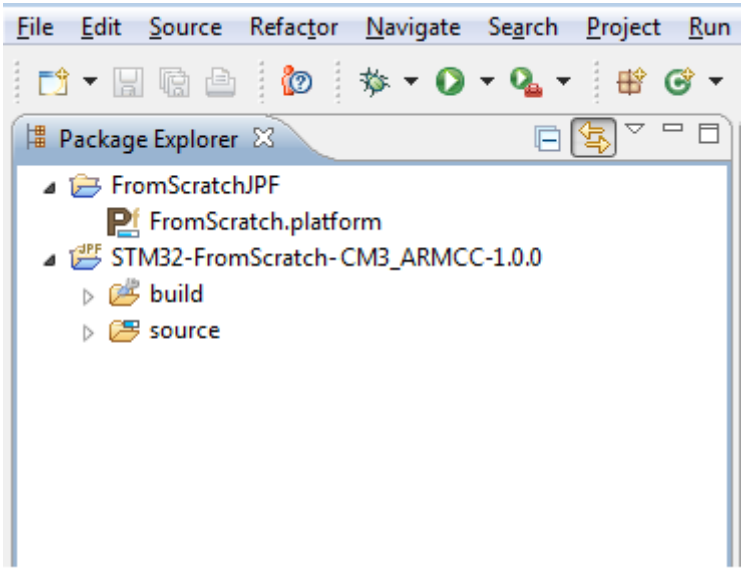

Figure 3.4. MicroEJ workspace

STM32-FromScratch-CM3_ARMCC-1.0.0 project has been created during the build operation ; it is notable by a JPF icon and contains JPF source.

[2]This name is created by concatenating the Device and Name entered in the dialog with the name of the toolchain being used.

3.2 Create the Java application that will test the JPF

3.2.1 Create the Java Project

Create a new MicroEJ Java Project. Do this by selecting : **File** → **New** → **Java Project**

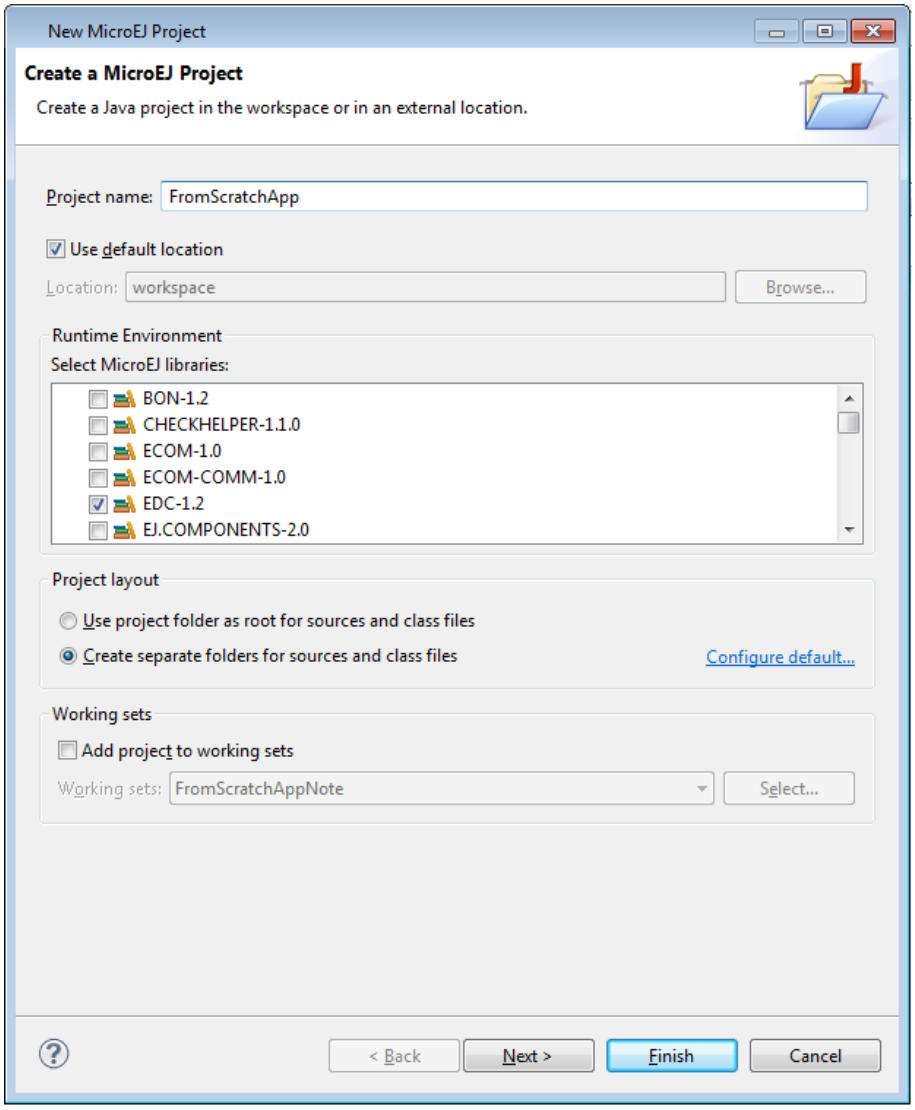

Figure 3.5. New Java Project

Enter the project name **FromScratchApp** and leave all the other settings unchanged. On pressing **Finish** a new project is created.

3.2.2 Create the main class

Right-click on the newly-created `FromScratchApp` project and select:

New → **Class**

Create a class called `FromScratch` in the package `com.is2t.example`, and tick the box asking for a main method to be created:

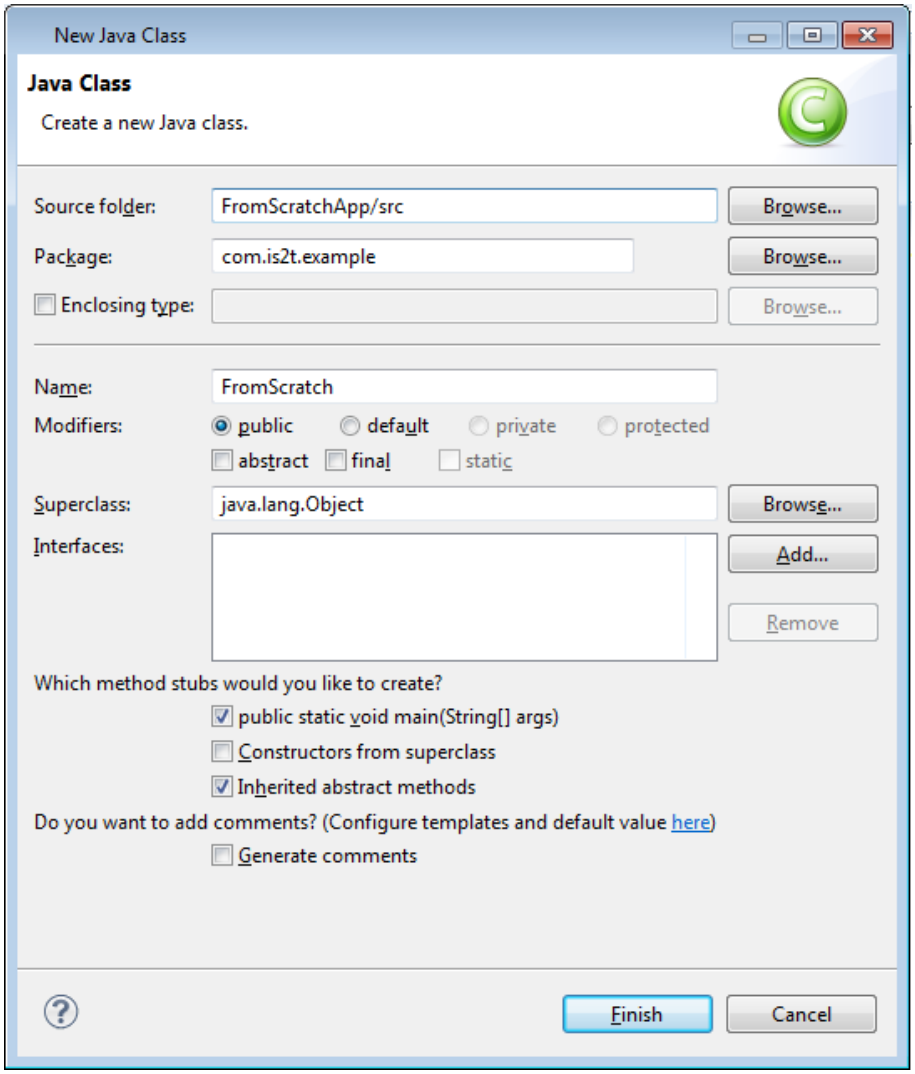

Figure 3.6. New Java Class

Fill in the Java code of the application, as shown below:

```
package com.is2t.example;
 public class FromScratch {

   public static void main(String[] args) {
    System.out.println("This is a simple application");
    for (int i = 0; i < 10; i++) {
     System.out.println(i);
     try {
      Thread.sleep(1000);
     } catch (InterruptedException ignore) {
      ignore.printStackTrace();
     }
    }
   }
 }
```

3.3 Build the Java application binary file

The next step is to build the Java application into a binary file that can be linked with the C parts of the platform. To build the application a suitable *launch configuration* must be created. Right-click in the Package Explorer on the Java class created in the previous step, and select: **Run As** → **Run Configu-**

rations... The **Run Configurations** dialog will open. Double-click the **MicroEJApplication** entry in the list to the left of the dialog to create a new configuration.

The details in the **Main** tab will be entered already, and should look like this:

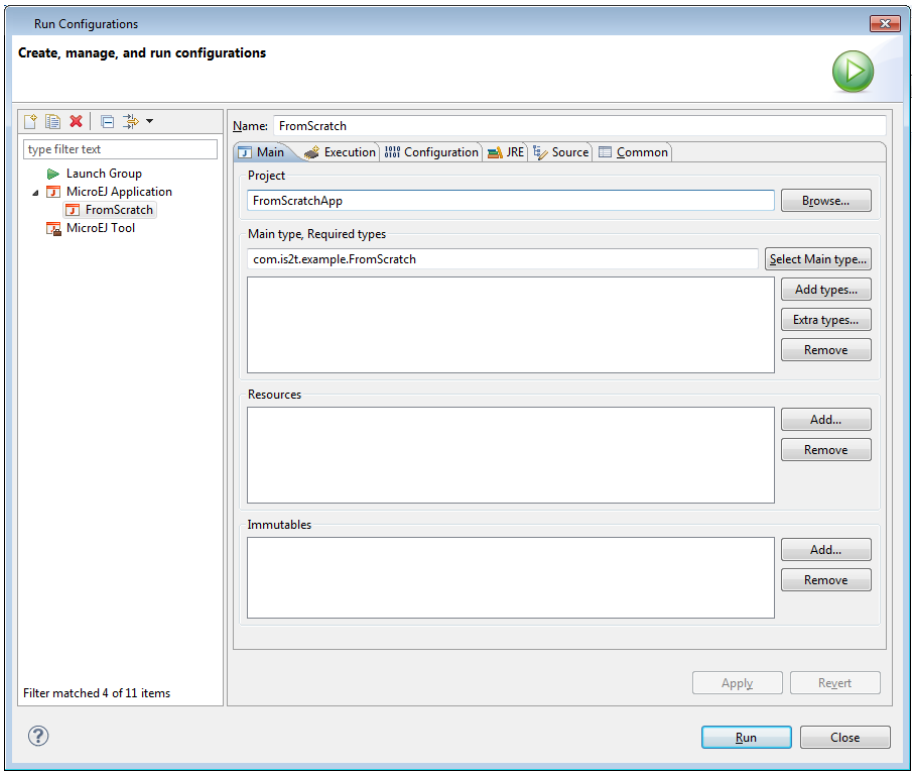

Figure 3.7. Run configuration - Main tab

Select the **Execution** tab. Check the **Execute on EmbJPF** option, as shown below:

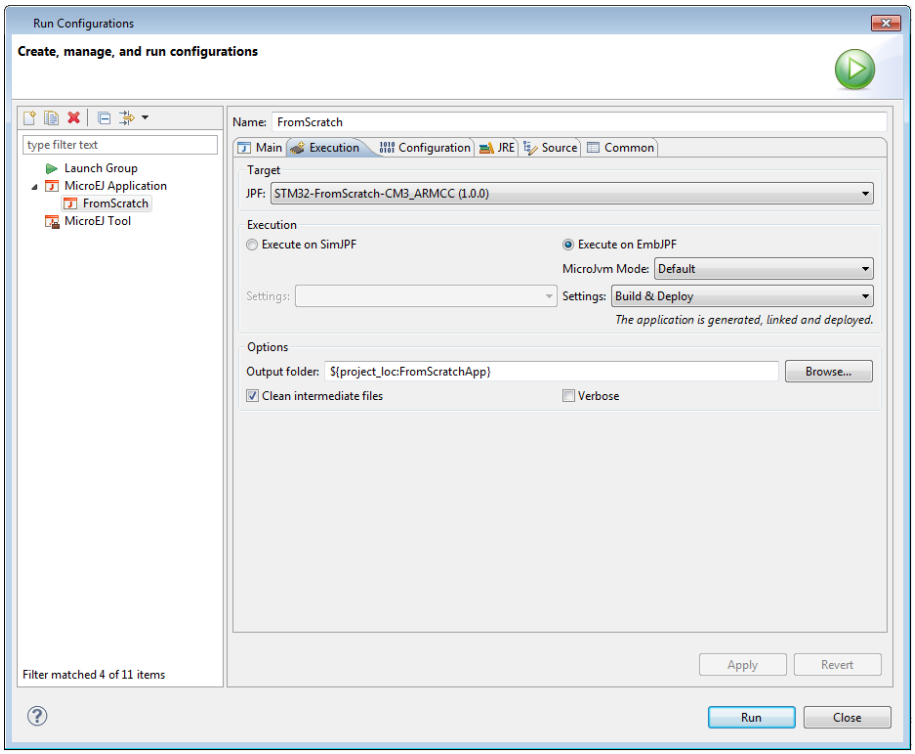

Figure 3.8. Run configuration - Execution tab

Press **Run**. The application is built into a binary file called `SOAR.o`, located in the `com.is2t.example.FromScratch` folder of the `FromScratchApp` project.

3.4 Create a file structure to hold the C code

The next step is to create the file structure for the C code project. This structure is based on the STM32Fxxx Standard Peripherals Library (SPL).

- Unzip the SPL into a suitable location, where it can be edited in-place – much of it can be deleted for our purposes.
- The `Project` folder of the library contains a template project folder, called `STM32Fxxx_StdPeriph_Template`, that will form the basis of the project for this example. Rename that folder to be called `FromScratch`. This will become the project folder for the C files.
- The template project contains support for several different C development environments. The files for use with Keil µVision are in the `MDK-ARM` folder; delete all the other folders.
- Delete all the files in the `FromScratch` folder – these are going to be replaced in the next step.

3.5 Add the MicroJvm virtual machine support files to the C project

The C project requires a number of files that provide platform-specific support to the MicroJvm virtual machine. A discussion of the content of these files is outside the scope of this Application Note. The required files are supplied with this Application Note. Two sets of files are provided; one set for the STM3220G-EVAL and another for the STM3240G-EVAL. Copy all the files from the relevant set into the `STM32Fxxx_StdPeriph_Lib_Vxxx/Project/FromScratch` folder (which will become the C project).

The Standard Peripherals Library structure should now look like this (the actual set of files may be different):

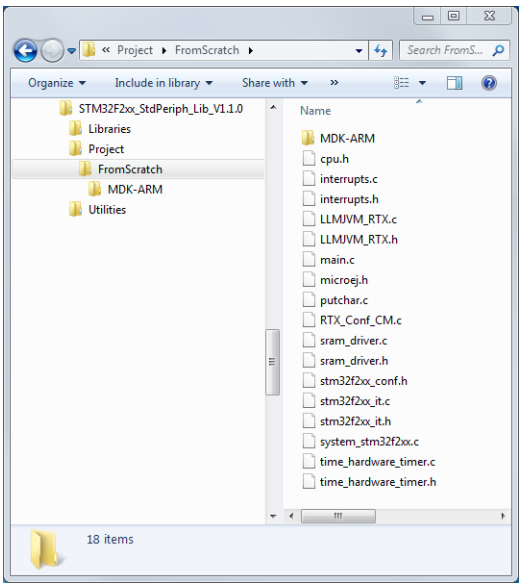

Figure 3.9. BSP files

3.6 Configure the Keil µVision project

The file `Project.uvproj` in the `MDK_ARM` folder is a Keil µVision project definition. It has been configured to suit the original contents of the template Standard Peripherals Library project and must be changed to suit the FromScratch requirements.

Double-click the `Project.uvproj` file to open it in the µVision environment. Several different configuration actions are required:

- Configure the project to use the RTX Kernel

  Right-click on the root of the project and select **Options for Target 'STM322xG_EVAL'...**. On the **Target** tab select **Operating system: RTX Kernel**.

- Configure the include paths

  Right-click on the root of the project and select **Options for Target 'STM322xG_EVAL'...**. On the **C/C++** tab, press the **...** button to the right of the **Include Paths** box. Press the **New (Insert)** button. Press the **...** button to the right of the new entry and navigate to the `source\include` folder of the `STM32-FromScratch-CM3_ARMCC-1.0.0` project (To find the location of this, in the MicroEJ environment, right click on the folder and select Properties). Press **OK**.

- Configure the langage level

  Right-click on the root of the project and select **Options for Target 'STM322xG_EVAL'...**. On the **C/C++** tab, set **Misc Controls** entry to "--c99".

- Add the extra C files

  Right-click on the **User** group of the project and select **Add Files to Group 'User'...**. Navigate to the `FromScratch` folder that holds the C files and select them all.

- Add the required libraries

  Right-click on the root of the project and select **Add Group...**. Name the group **MicroEJ**.

  Right-click on the new group, and select **Add Files to Group 'MicroEJ'...**. Navigate to the `com.is2t.example.FromScratch` folder of the FromScratchApp application, and select the `SOAR.o` file.

  Right-click on the new group again, and select **Add Files to Group 'MicroEJ'...**. Navigate to the `source\lib` folder of the `STM32-FromScratch-CM3_ARMCC-1.0.0` project, and select the `javaruntime.lib` file.

The project should now look like this:

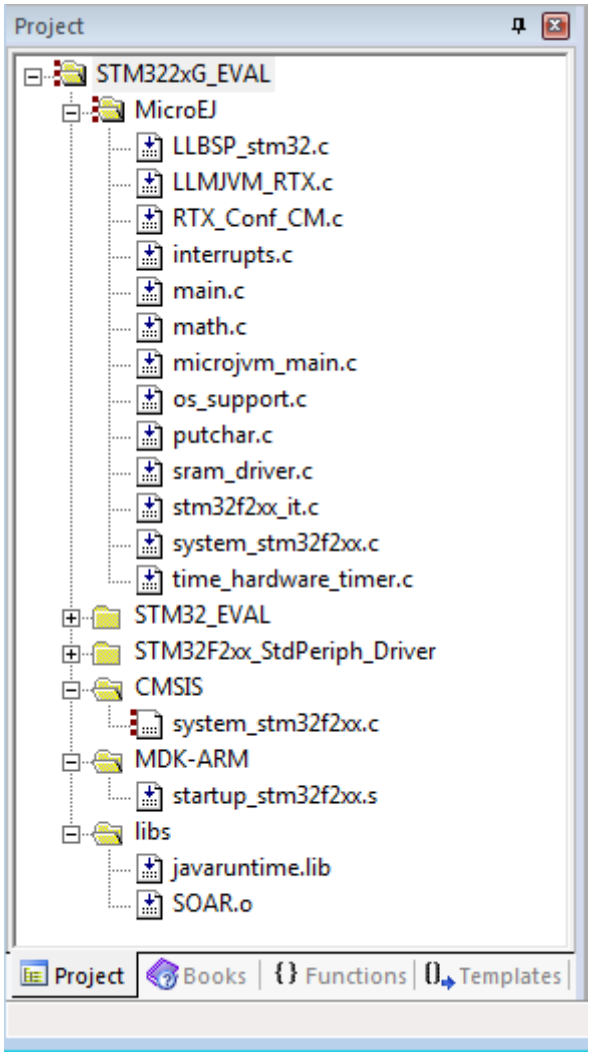

Figure 3.10. µVision Project Content After Configuration

3.7 Build and deploy the C project

The C code should now compile cleanly. Build it using **Project** → **Build target** (**F7**). It can now be downloaded to the target board using the ST-LINK/V2 connection. Reset the board to run the application, and the output will appear in the terminal emulator connected to the serial port.

4 Document History

Date	Revision	Description
February 19th, 2013	A	First release
September 2nd, 2013	B	Change in document generation
November 12th, 2013	C	MicroEJ ARM Cortex-M (Keil MDK-ARM) 2.0.0 compatibility
April 29th, 2014	D	Update BSP download links
June 5th, 2014	E	MicroEJ ARM Cortex-M (Keil MDK-ARM) 3.0.0 compatibility
October 1st, 2014	F	MicroEJ 3.1.0 compatibility

Headquarters
11, rue du chemin Rouge
44373 Nantes Cedex 3
FRANCE
Phone: +33 2 40 18 04 96
www.is2t.com